IA-SIG Newsletter
"The Interactive Audio Journal"
www.iasig.org

****

****

In This Issue:

Section I: From the Chairman — Some opening comments from IA-SIG Chairman, Mark Miller

Section II: Official IA-SIG Announcements — All official announcements regarding IA-SIG members, activities, and special events. In this Issues, IMX and the upcoming Project BBQ.

Section III: Working Group Reports — Status Reports from the 3D Working Group, the Interactive Composition Working Group, The Audio Advisory working Group, and the Multi Format working Group.

Section IV: Features — This section contains features and columns on varied topics of IA-SIG interest. This issue features Part 1 'A Roadmap' (which represents one very big idea) from Chris Grigg and interviews with Jim Hedges and the Soul Reaver sound team at Eidos USA.

Section V: Industry Corner — This section contains news, interviews and information from our member companies. In this issue we have our second "Engine Roundup" segment in which we share conversations with and comments from QSound's Scott Willing and Microsoft's Brian Schmidt on the state of their 3D Audio technologies. *<Ed. Note: Items in the Industry Corner section are provided as useful information but do not necessarily represent the views of the IA-SIG or its management.>*

Section VI: Developer's Corner — At last, a place where the developers can speak their minds. The interactive audio industry is bursting with men and women ready to both complain and cheer about various aspects of creating content. Here is our outlet for them. This issue features tales from the first place where video games found mass public appeal: the arcade!

If you are interested in contributing to "The Interactive Audio Journal" please contact Mark Miller (mark@harmonixmusic.com)

****

Please Join the IA-SIG!

The Interactive Audio Special Interest Group (IA-SIG) exists to allow developers of audio software, hardware, and content to freely exchange ideas about "interactive audio". The goal of the group is to improve the performance of interactive applications by influencing hardware and software design, as well as leveraging the combined skills of the audio community to make better tools.

The IA-SIG has been influential in the development of audio standards, features, and APIs for Microsoft Windows and other platforms, and has helped numerous hardware companies define their directions for the future. Anyone with a commercial interest in multimedia audio is encouraged to become a member of the IA-SIG and participate in IA-SIG discussions. (Visit http://www.iasig.org for membership details).

****

****

IA-SIG Glossary of Terms

• "DLS" — DownLoadable Sounds, this is a standard created by the IA-SIG to allow users of computer audio cards and software synthesizers to create their own instruments and sounds rather than be limited to a specific sound set.

• "WG" — Working Group, the working end of IA-SIG, these groups represent various causes in the interactive audio industry. They work towards goals that might not otherwise be possible in any other body.

• "GDC" — Game Developers Conference, held once each year, this conference brings together some of the best in the game industry for both computers and game console systems to discuss current issues, teach classes, and explore new opportunities for gaming.

****

Section I: From the Chairman
============================
"Q3 1999"

It's been another intensely active period for the IA-SIG. The past three months have seen more activity than at any other time in our history. Among our recent accomplishments, we can list the following:

- Tremendous progress by two of our Working Groups (The 3D Working Group and the Multi Format Working Group)
- The Interactive Composition Working Group revitalized
- Two new Working Groups (The DSPWG and the IPWG) formed and launched
- The Web Site re-designed and re-launched
- The second issue of the news letter published

More specifically:

With the 3DWG's pending finalization of I3DL2, the world will final hear interactive acoustical environment modeling (that's reverb, in plain English) on PCs and other interactive entertainment platforms. With the pending completion of the MFWG's mission, the ins and outs of multi channel audio systems should finally make some sense.

The new ICWG Chairman, Brad Fuller, with the help of Vice Chairmen Jim Hedges and David Javelosa have set the group on fire. I have to say that the recent conversations on the nature of Adaptive Audio Systems that have been blazing on this list for about the past month have been the most interesting and stimulating discourse on the topic that I think I have ever heard. If I were you, I would get on this list and get involved.

I also expect to see the launch of two new Working Groups within the next week or so. The first is the Digital Signal Processing Working Group (DSPWG). Their stated goal is to 'document, specify, standardize, and popularize an architecture and API which will allow applications to send and receive back real-time streams of audio to a DSP Filter for processing'. The second is the Intellectual Property Working Group (IPWG). The IPWG is being formed to address the issues surrounding the reasonable use and licensing of sound sets in interactive media.

More information on all of the IA-SIG working Groups can be found on our recently redesigned Website (http://www,iasig.org). As before, we remain committed to keeping the site stocked with the most up to date news information on all aspects of the interactive audio community.

In the near future, look for new initiatives from the IA-SIG in the area of Internet Audio.

Finally, I offer you this, our second news letter. I hope you find it as interesting to read as it was to edit.

Yours,

Mark Miller
Chairman
****

Section II: Official Announcements
================================

The BBQ Topic for '99

"INFLUENCING HARDWARE AND SOFTWARE FOR MUSIC
ON COMPUTERS OVER THE NEXT 5 YEARS"

Austin, Texas;

This October 14th-17th, 50 of the brightest minds in the computer industry will once again trade the comforts and chaos of corporate America to converge on a secluded 360-acre ranch in Texas without TV's, phones or computers to discuss, argue, debate, brainstorm and possibly alter
the course of history for music on computers over the next five years. The Fourth Annual Texas Interactive Music Conference and BBQ is preparing the next great assault against the toughest questions facing our industry today on the topic: "Influencing hardware and software for music on computers over the next five years." It is the only independent conference for the computer audio industry and is hosted by multimedia guru George Sanger, better known as The Fat Man.

But don't let the name fool you, Project BBQ is one of the most prestigious, influential and beneficial networking events of the year. This unusual conference gathers the movers and shakers of the audio industry together for a long weekend of concentrated, uninterrupted discussions, debates and roundtables, as well as top-notch panel and speaker sessions with industry leaders. A well thought-out and run program puts attendees to work within thirty minutes of arrival at the ranch on Thursday and keeps them on a tight schedule until their shuttle departs Sunday afternoon. Sessions run from 8am until 8pm and informally continue with networking through dinner and usually well into the night.

Both Mark Miller, the IA-SIG Chairman, and Tom White, the President and CEO of the MMA serve on the BBQ's Board of Technical Advisors.

For inquiries regarding Project Bar-B-Q, relating to its business, administration, and inimitable cowboy style, email: spanki@outer.net or visit http://www.fatman.com/bbq9901.htm

<u>The IA-SIG and the MMA at IMX</u>

As a part of its on going sponsorship and support of the Interactive Music Xpo(IMX), the IA-SIG and the MMA will present two conference sessions at this year's show:

Tom White, President and CEO of the MIDI Manufacturers Association will present 'New Technologies from the Manufacturers', on Monday August 9th from 4:00 to 5:00 PM in Room 1E15. Panelists include: Korg USA: Michael Kovins, President, Van Koevering Company: David Van Koevering, CEO, Dr. Robert Moog.

Mark Miller, the IA-SIG chairman will present Music and Gaming on Tuesday, August 10th from 11:15AM to 12:15 PM in room 1E13. Panelists include: Microsoft: Steve Ball, Segasoft/Heat.net: Gordon Lyon, Tommy Tallarico Studios: Tommy Tallarico, Electronic Arts: Rob Hubbard, Whitmoreland Productions: Guy Whitmore, Owner.

The Interactive Music Xpo(IMX) continues to establish itself as the premier music industry event. Support from major vendors, associations, and industry luminaries, along with the announcement of a Benefit Concert for the Aubrey Fund for Pediatric Cancer Research (part of the Memorial Sloan-Kettering Cancer Center), enables IMX to serve all levels of the musical community.

IMX Sponsors include: a2b music, Microsoft, Lucent Technologies, Sonic Foundry, MCY Music World, Inc., VIBE, SPIN & BLAZE Magazines, RIAA, NARAS, ASCAP, Midi Manufacturers Association & Interactive Audio - SIG, Electronic Musician Magazine, Gig Magazine, J-Bird Records, Hotz Interactive Exhibitors & Speakers Include: KORG USA, Diamond Multimedia, Hilary Rosen (CEO - RIAA), Todd Rundgren, Phil Ramone (Producer), Liquid Audio, John Lennon Songwriting Contest, Cakewalk, Digidesign, Tunes.com, Tony Visconti (Producer), John Alagia (Producer - Dave Matthews Band), Seer Systems, CDKnet, CustomDisc.com, EZCD.com, Tune 1000/Vorton Technologies, DiscMakers, GoodNoise, Beatnik, Mixman Technologies, HyperLOCK, The Orchard, Kellar Bass Systems... and many more!!!

IMX is scheduled for August 9 & 10, 1999 at the Jacob Javits Convention Center in New York City. Please visit http://www.imusicxpo.com for details on updates.

<u>"The IA-SIG re-launches the Interactive Composition Working Group (ICWG)"</u>

The Interactive Composition Working Group (ICWG) is now officially in it's 2nd major phase of operation, with revised goals and a management group that is dedicated to making adaptive audio systems a reality for all interactive products - on all platforms. In the coming months, the group will be putting their heads together to develop the frameworks for adaptive audio systems that will propel audio in interactive products to entirely new levels.

The group is currently seeking talented individuals to assist them in accomplish these worthy goals. If you're not currently part of the ICWG, Brad Fuller, the group's new chairman, would like to invite you to join and contribute to this exciting initiative. More information on the ICWG can be found in section III of this issue and at http://www.iasig.org under 'Working Groups'.

<u>"New Working Groups Proposed"</u>

DSPWG — The Digital Signal Processing Working Group has been proposed under the leadership of Keith Weiner of DiamondWare. Rob Hubbard, the group's designated Steering Committee Representative offered the following invitation to IA-SIG members last week:

"Currently, there are many software and hardware technologies that capable of doing various DSP type of functions, such as filters, flangers, choruses and reverbs. Unfortunately, for the game developer there is no standard way to use any of these capabilities. Traditional media such as film and TV, have access

to very expensive state of the art DSP processing units, but in games we still, for example, don't have a way to add a simple low pass filter to a wave sample being played by DirectSound".

> *<Ed. Note: IA-SIG efforts are intended to be platform and hardware agnostic wherever possible. DirectSound and the Microsoft Windows platform are stated here as examples only because they are familiar reference points for most readers.>*

The purpose of the DSPWG is to further explore what is currently available (write it yourself !), and try to come up with an API that can address current shortcomings on the PC and other platforms. The benefits of having this API available are numerous — better quality real time interactive sound on games, creating more work for the industry, and an opportunity to get hardware vendors involved and perhaps come up with a way for them to provide DSP acceleration, creating a business opportunity for them.

For additional information, please contact Keith <keith@dw.com <mailto:keith@dw.com>>, or see our Web Site under 'Working Groups'

IPWG — The IA-SIG's Intellectual Property Working Group is expected to launch by August 1, under the leadership of Mark Miller. Mark outlines the purpose and goal of the new Working Group as follows:

PURPOSE: The purpose of the IPWG is to facilitate and insure the continued development, availability, growth, and profitability associated with the marketing, distribution, and licensing of sounds and sound sets to the interactive audio community.

GOAL: The immediate goals of the IPWG are to facilitate improvements in interactive audio and to make use and licensing term recommendations regarding reasonable and appropriate distribution of sounds and sound sets into interactive media and playback environments. The recommendations will be based on broad industry input and be consistent with the IPWG Purpose above and copyright-owner rights as intended under existing and evolving copyright laws.

For additional information, please contact Mark <Mark@harmonixmusic.com>, or see our Web Site under 'Working Groups'

****

Section III: Working Group Reports
==========================

The Working Group is the main functional aspect of the IA-SIG. Individual Working Groups form around specific issues of concern within the industry. The members meet in person or via the Internet and work towards developing technical specifications or recommended practice documents. These documents represent industry consensus and are published and made available to all interested parties.

Working Groups (WGs) report progress quarterly, with the most recent reports being published in each IA-SIG newsletter. Summaries of WG functions and activities are maintained on our website.

3D AUDIO WORKING GROUP (3DWG)
Chairman: Conrad Maxwell, Conexant <conrad.maxwell@conexant.com>

The 3D working group has been busy "dotting the i's and crossing the t's" of I3DL2 (Interactive 3D Audio Guidelines, Level 2). The I3DL2 preliminary document was approved just prior to the Game Developers Conference in March. At press time, the I3DL2 document is undergoing final review by the 3D Working Group members.

The I3DL2 document details the 3DWG's recommended approach to providing state-of-the art Interactive 3D audio using a pre-set reverberation to simulate a common environment for sound source and listener. Occlusion and obstruction effects are also specified. An example API implementation (two 'C' header files) is included in the specification. enumerating a recommended "DirectSound Property Set" implementation of the low level I3DL2 source and listener controls. The Property Set also includes a number of useful presets for rooms and material types for occlusions and obstructions.

It is expected the 3D working will take up the issue of 3D MIDI specification as a future (and possibly its next) work item.

Finally, the issue of effects beyond 3D were discussed as a possible next item. Given the current I3DL2 specification, it was noted that some of the problems solved could be easily applied to effects like flange, chorus and other studio effects. Currently this is beyond the charter of this working group. However, many members expressed interest in this topic. It is expected that many current 3DWG members would participate in such a workgroup, were it to be formed.

Submitted by Brian Schmidt <bschmidt@microsoft.com>
3DWG Steering Committee Representative

AUDIO ADVISORY WORKING GROUP (AAWG)
Chairman: Scott McNeese, VLSI <scott.mcneese@tempe.vlsi.com >

The AAWG is currently engaged in a re-evaluation of its goals and structure. The re-evaluation is expected to be completed by September and the results published in our next issue. At this time, there are no other specifics to report.

Submitted by Mark Miller <mark@harmonixmusic.com>
IA-SIG Chairman

INTERACTIVE COMPOSITION WORKING GROUP (ICWG)
Chairman: Brad Fuller <bfuller@pacbell.net

The ICWG is alive and it is kicking-a**! In May we established goals and an aggressive agenda to help us meet those goals. But, if you haven't been paying attention to the ICWG, you've been missin' some fascinating discussions.

First, let me review the goal that we established in May: "Establish and Develop Voluntary Technical Guidelines for Adaptive Audio Architectures." Common questions that you may be asking yourself:

Q: What is Adaptive Audio?
A: Adaptive Audio is audio that is delivered via a system that allows for direct or indirect control of the data and/or the data stream.

Q: What is the ICWG's definition of an "Architecture"?
A: Architecture: a collection of components and interactions among those components. A description of elements from which systems are built, the interactions among those elements, the patterns that guide their composition, and the constraints on these patterns.

Q: What is the desired output or result of the ICWG?
A: We hope to build a common lexicon of adaptive and interactive music terms by the end of the year. This is an important step as it will give us a foundation to move discussions and decisions along. More importantly, we hope to have the first draft of the Adaptive Audio Architecture completed by CGDC 2000.

Now that we have that out of the way, I'm happy to report that there's been a flurry of activity on the ICWG reflector to help us meet these goals. One of our first threads of discussion was on intellectual property and how, when, and if the IA-SIG should be involved. Although the subject of the discussion focused on the topic of Adaptive Audio Engines, the arguments that followed are applicable to other specialties of the IA-SIG. If you are faced with IP issues, you might ask an ICWG member or visit our reflector for info on our discussions.

What really got the ball rolling for us in June was the 'Genesis' thread that brought forth two topics:

1. It emphasized the difference (and hence paradigm-shift for developers and users) between the linear audio world of Film/TV and audio developed for interactive products.
2. It defined Adaptive Audio System (AAS) categories. While there was virtually no disagreement with the AAS categories, the inevitable religious wars erupted. This included debates regarding term definitions all they way to internal specifications AASs. Our discussions resulted in great ideas, and I must say a very professional discourse among the ICWG participants.

We are looking forward to a very productive year. We certainly have the skills among the members to meet our goals for this year and before the CGDC 2000 convention.

Submitted by Brad Fuller

MULTI FORMAT WORKING GROUP (MFWG)
Chairman: Michael Land, LucasArts <mland@lucasarts.com

The MFWG is reaching the wrap-up stage of its proceedings. We've established the set of input and output formats that we are including in our recommendations, and the preferred mappings for connecting channels from input formats to channels in output formats. There are currently two last issues to resolve before we conclude. The first is that we are improving our original table-based approach for specifying mappings by finding patterns and presenting the mappings in a more easily understood rule-based format. The second is that we are evaluating some systems that are currently in development and comparing their real-world implementations to our more theoretical ones (it remains to be seen whether or not this evaluation will have any effect on the final form of our recommendations). We should probably be wrapping up this workgroup and presenting our recommendations within 2-3 weeks.

Submitted by Michael Land

****

Section IV. COLUMNS
This section contains features and columns on varied topics of IA-SIG interest.
==

Part 1 of 2 – For IA-SIG Newsletter

A Roadmap

Chris Grigg, Control-G

chrisg@control-g.com

Purpose

In the weeks after the excellent Game Developer's Conference 1999, a funny thing happened inside my head. Several separate threads of thought which I had previously regarded as not very related unexpectedly converged into something strangely significant-feeling. Late one night I sat down to write about it, and what came out was a new long-range view of the relationship between traditional software development practices, certain emerging software technologies, and the under-met needs of interactive audio content creators. GDC made it clear that we all agree on what the problems are, and that we for the most part agree on the kinds of things we want the content creators to be able to do. The purpose of this piece is to suggest a conceptual 'roadmap' that may be able to help us all get there much, much faster. The concepts involve reconsidering how units of audio software are organized, and how they communicate with one another.

While there's ample room for disagreement about any of the implementation details I suggest here, I hope this document sparks a public conversation that leads to a shared vision that we can all benefit from. Just think how much progress could be made if all our considerable energies could somehow be coordinated.

Audience

This article is an adaptation of that night's work, addressed to IA-SIG's broad audience of audio technologists and content creators. Because the subject matter veers back and forth between software technologies and media content creation, programmers may find it irritatingly imprecise at the same time that soundmakers find it unnecessarily technical; unfortunately, this probably can't be helped given the interdisciplinary nature of the point I'm trying to make.

Motivation

In any given area of technology development there comes a stage at which the focus shifts away from early struggles (i.e. getting the basic function to work at all), and toward finding the best, generalized way to control the thing over the long term. Think of cars, airplanes, computer OS's, networks, etc. – eventually a stable, standard form of control emerges (wheel & pedals, GUI, WWW). With IA-SIG developments such as the work of the Interactive Composition Working Group and now a DSP plug-in WG in the works, that same transition for game sound now appears to be on the horizon.

If the groundwork we lay in these initiatives is too restrictive, it will take years to extricate ourselves and retool. Doing this really right the first time, on the other hand, could mean rapid movement towards solving some of the biggest and oldest problems that game developers and publishers face: integrated authoring tools, sound designer-

controlled audio interactivity, and platform-independent audio content. And audio API and hardware vendors will also benefit from new kinds of business opportunities. The real point, of course, is the leap forward in audio experience quality that should follow.

Thesis

My thesis is that the reason why we've been so stuck here for so long is that we have a bottleneck. It's called "programming".

Today's reality is that to get a game to make any reasonable sound, a developer has to choose somebody's audio system and then write code that calls its functions.

There's only one thing wrong with that: Everybody's audio system furnishes a completely different set of functions; further, each one requires the developer to call those functions in a completely different way.

There's only one thing wrong with that: Once you've picked an audio system and written your game engine to use it, it's prohibitively expensive to change audio systems. (It happens, but it's rare.)

There are only three things wrong with that: Nobody's audio system does every single part of the job really well; nobody's audio system runs on all the platforms to which the developer wants to port; and everybody's audio system needs its own special auditioning/testbed applications.

In other words, today's large audio systems tend to be all-inclusive solutions that succeed really well on a single platform; but their functionality is hard to customize; they aren't transportable; and they require custom tools (and sometimes custom media formats).

The funny thing is, each of these big audio systems is already built up internally from many separately useful classes and functions that could do most all the things you'd ever want to do, if only you could somehow rearrange them. A huge number of implementation differences make that observation practically irrelevant given the current way of doing things, but: What if you *weren't* stuck with having to pick a single-vendor solution? What if (for example) you *could* use vendor A's great synth with vendor B's great 3D mixer? What if vendor C could write a MIDI-controlled DSP module to plug in between the synth and the mixer? What if you could set up that connection yourself, and then drive all three pieces from a scripting engine?

What if you could design the exact audio system you need, without a programmer hand-writing code to deal with buffers and property sets, by drawing block diagrams?:
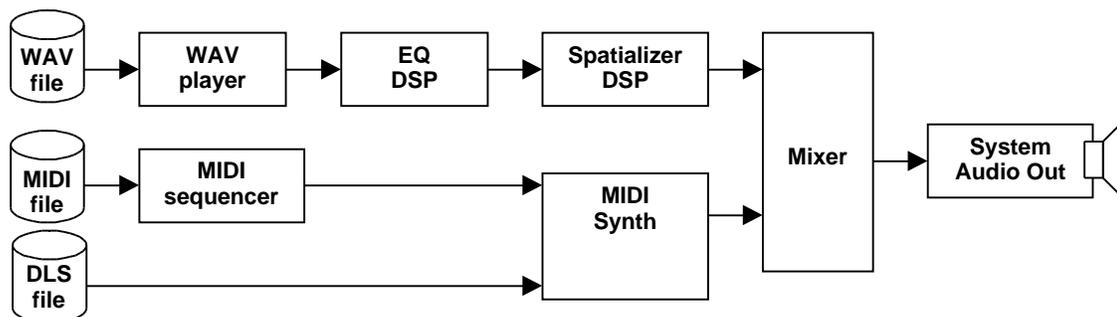
In this 'What if?' world, the units of software organization get smaller, more modular; each software unit takes on a more specific function; and all of the software units have to be able to talk to one another. This way of structuring the audio system is fundamentally different from our present practice of developing and using large-scale, do-everything audio API's.

To make it possible to organize our runtime software and authoring systems along these lines, we would need three main things:

- A standardized software component framework with an object model
- A standard for audio signal interconnection between software components
- A standard for control signal interconnection between software components

Each of these things is good, but it's the flexibility of the combination of the three that really gets us where we want to go.

Unfortunately, this article is too long to fit into a single issue of the Newsletter, so I'm going to have to save the control signal interconnection discussion (which is long) for next time. The remainder of this installment discusses the requirements and benefits of the first two, and winds up by offering a set of suggested design principles to observe in implementing this new world. Along the way I'll point out the synergistic benefits that follow from doing these things in combination.

Note that few (or none!) of these particular ideas are new; the only thing that's new here is their combination and application to game sound, and the recognition of their advantages for our specialized context.

A Standardized Software Component Framework with Object Model

If we're going to be assembling small software components, we'll need to have a way to specify which components we want to use in a given context, and we'll need a way to make sure the right parts (functions, methods) of each component get called at the appropriate times. These sorts of services are usually provided to software components by an underlying software architecture called a framework. You can think of the framework as the thing that all the plug-ins plug into. A game would include a framework to support its audio system.

Although it's conventional to furnish different kinds of frameworks for different kinds of functionality (i.e. you could decide to offer one interface for DSP plug-ins, and a separate interface for, say, music synth plug-ins), I would like to suggest that there are excellent reasons to treat all software components in this world exactly the same, and to plug every component in a given audio system into the same framework. Placing components in a framework would be done in a GUI authoring tool – you'd drag boxes around to form diagrams looking pretty much like the one above.

That's because in this world, <u>everything</u> is a plug-in. All components use the exact same low-level calling interface (set of function/method signatures); a standard mechanism allows all components a way for each one to express its unique capabilities and unique set of controls.

Managing a framework that supports this sort of arbitrary placement of an arbitrary set of software components within it in a general way is a specialized job best addressed by use of an 'object model'. An 'object model' is an abstract description of the various pieces that comprise a given system, usually expressed in a way that harmonizes with object-oriented programming concepts like class, object, and property. For example, browsers use a 'document object model' that describes all the possible entities in a webpage rendering environment. Another example is AppleScript, an object-oriented application scripting language whose object model allows (among other things) a way of reporting what documents are available, and a way of manipulating the properties of any desired document. The point to remember here is that an object model provides a single, standard mechanism for finding out what's going on with the components inside the framework, and for controlling them.

It's possible that our framework might need to be implemented in pretty different ways for different host OS environments, but its object model and the binary form of its expression would remain the same in all cases, for reasons that should become clear soon.

When a framework is able to understand a software component object model, the selection and arrangement of components present within it can be set, queried, and altered by other pieces of software, such as the game code. When our runtime audio system is assembled in a framework with an object model, the audio architecture can be adaptively customized for the purpose at hand. If we were to drive the framework's object model interface with a scripting engine, then sound content creators would be able to configure the audio system on their own, without programmer assistance.

Note that although a native implementation of the framework would need to be built for every platform we want to run on, the object model itself is a purely logical construct. So, given a similar set of available audio software components, any audio system configuration script you might write could conceivably work without without modification on any platform where the framework's been implemented.

Technical Aspects
None of the above makes any assumptions about what software component technology would be best to use when implementing the framework. The standard component architectures (COM, SOM, CORBA) may or may not be the best choices; obviously it makes sense to take advantage of any given platform's strong-point technologies.

Note that as long as the semantic structure of the framework's low-level component interface was the same (i.e. used the same method or message names), any available and sufficiently efficient component technology could be used; in fact, a different component technology could be used on every platform, and audio content authored for one platform would still work without modification on any other platform.

Interconnection
Software component frameworks *per se* are nothing new; for example, ActiveX is a software component framework. But we're talking about a framework intended specifically for the purpose of realtime media work. Making an actual audio system out of the isolated software components we've plugged into our framework requires signal

interconnections between the components. For example, a realtime EQ plug-in needs to be fed a stream of digital audio, and the EQ's output needs to be fed to some other component to be heard.

Connecting signals between the software components would also be done in the same GUI tool used to place the components. Each component box would have its own set of input and output ports as required by its purpose, and you'd draw patchcords to connect them. This configuration editor tool at first glance would probably look a lot like Opcode/IRCAM's MAX.

In an ideally simple world, differences among various kinds of signals (audio, MIDI stream, command stream, etc.) would be invisible; but I'll describe audio signals and control signals separately so that in the second installment of this article I can get deeply into how it might be possible to accommodate the unpredictably unique functions of all possible classes of software component in a uniform way, and why that could be a really good idea.

A Standard for Audio Signal Interconnection Between Software Components

We'll need to define a standard for audio signal interconnection between components, because the framework is going to need to know what we want to happen when we tell it to 'Connect the output of the WAV file player to the input of the EQ.'

I frequently hear the concern that permitting arbitrary patching between digital audio functions is going to require massive duplication of large sound buffers on a continual basis, leading to catastrophic failure; however, I just don't see it. (I refer here to actual RAM audio buffers, not any particular existing audio API's buffer abstraction objects.) You don't move the data between buffers; instead, at every audio interrupt (or whatever your clocking mechanism is) you control the order in which the various components perform their operations on the buffers. The framework could use just one buffer per signal chain, not one buffer per component output or input.

In the WAV/EQ example, the framework would know from the direction of the patch that at every audio interrupt (or whatever) it needs to first get a bufferfull of new audio from the WAV player, and then call the EQ to process the buffer; presumably the EQ's output is feeding a mixer or other component, so that downstream component would be called next. Time-domain multiplexing each buffer, if you will. Buffer duplication only becomes necessary when the audio signal path forks (which is relatively rare).

In other words, intelligence in the framework would translate this signal path configuration:

| WAV player | → | EQ DSP | → | Spatializer DSP | → | Mixer |
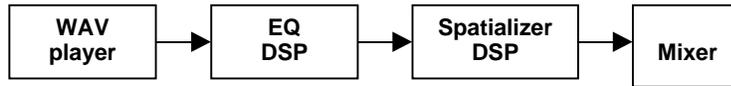
into one single RAM audio buffer and a function call sequence (at each audio interrupt or whatever) something like this:

| Path1Buffer |

```
wavPlayer->UpdateAudio( &Path1Buffer );

eqDSP->UpdateAudio( &Path1Buffer );

spatializerDSP->UpdateAudio( &Path1Buffer );

mixer->UpdateAudio( &Path1Buffer );
```

Settling on a standard of one audio buffer per unforked signal path would give the framework all it needs to make audio connections between components, and be space- and time-efficient. Note that this places very few constraints on the system: at any given time all buffer sizes would have to be the same size, but different platforms could use different buffer sizes, and in fact the same game could use different buffer sizes at different times if desired.

Interconnections in the Framework's Object Model

So how would the framework know about these signal connections between components? It would keep track of them in terms of its object model.

Consider our block diagram again. The idea of the object model is to make every box in this kind of block signal flow diagram a **component**, and to make every arrow a signal flow **connection**. The object model is then all about the abstract concepts of components and connections. When the framework is written, it will be in an object-oriented programming language, and it will almost certainly spend a lot of its time manipulating objects of class `component`, and objects of class `connection`.
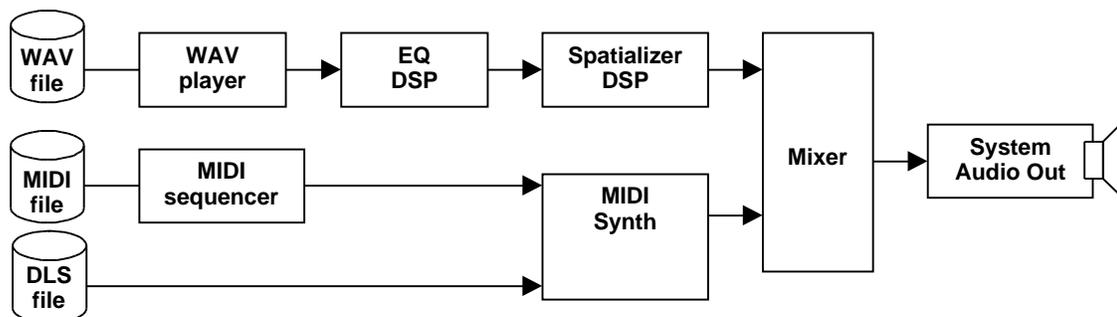
In the largest sense, a component is anything that creates, processes, or receives a signal. Media (the disk-cans in the diagrams) gets turned into signals by components; other components synthesize signals on their own, no media required. A connection is whatever it takes to plug a component output into another component's input.

To support this way of thinking– and to get to all the benefits that follow from it (enumerated at length below and in part two of this article, next issue)– the framework and object model would need to furnish a way to introduce components (typically a component would be a low-level audio API), a way to construct configurations of component instances, and a way to make connections between those component instances. Ideally, components' maintenance needs would be automatically met completely within this same architecture, but components' unpredictable and unique requirements may make that impractical.

Benefits I
At the cost of implementing a framework with object model for each desired OS/platform, taking a components-and-signal-flow view of the world has several benefits:

1. **Promotes Programmer-Sound Creator Understanding.** It gets the programmers talking the same language as the sound content creators, which is likely to help increase understanding (not to mention civility), tending to improve delivered game sound quality. This interconnection model already makes sense to audio content creators from their studio experience and, given a good framework implementation, should be preferred by many game engine programmers too.

2. **Increases Clarity.** It also gets the programmers thinking in terms of the abstract audio processes operating on audio signals, rather than any particular OS or API vendor's low-level programming model. With well-factored audio DSP components, this way of thinking will lead to a better understanding of the dynamic CPU/DSP expense of various operations. (Doing the architecture right and in open-source would be a way to minimize adoptance reluctance.)

3. **Simplifies Writing of Audio Code.** With a manager layer in place to handle the ordering of calls on buffers and the relationships between API's, the coder just doesn't have to do as much work, which is always welcome.

4. **Modular Interface Promotes Use of More Audio API's.** Once the framework's been learned, adding further audio functionality becomes easy– for example, use of specialized EQ's or inline effects processors could become common. As this area grows, vendors of DSP now available primarily as professional plug-ins may start to move into licensing for the game runtime environment with SDK-style versions of their API's packaged for the framework. One can easily imagine the creation of 'mini-DSP-slice' libraries with multiple versions optimized for various purposes. More vendors get to play.

5. **Encourages a View of Audio Libraries as Interchangeable.** By making it easier to see the different API's as the audio components they are, and by making it easier to swap those components, it will make developers feel less locked-in to

particular vendors and particular API's. This should in turn drive competition and improvement in the low-level audio components as developers swap and compare more frequently.

6. **Simplifies Porting of Game Engines and Titles.** The same component-and-connection framework could be implemented on any modern OS, using the same high-level interface across platforms. That means that the audio part of the job of porting a game or engine to any platform where the framework has been implemented would be much simpler. (Note that the architecture only has to be implemented once per platform for <u>all</u> developers' content and game engines to port. Note also that for hardware vendors, seating the low-level API functionality primarily in the card drivers rather than in host CPU code could make deployment to other platforms using the same hardware profile [i.e. PCI/PCMCIA] much easier by reducing the amount of middleware that needs to be written per platform.)

7. **Encourages Porting of Low-Level Audio APIs to New Platforms.** Once the framework exists on a platform, then API vendors have the basis for a built-in market on a new platform as developers begin to port titles to that platform within the framework.

Design Principles

In the next issue I'll get into the important area of control signals and their routing, but for now let me close by moving away from details and zooming back out to the long view. The point of this whole media component interconection idea is to meet all of our needs in the broadest possible way. For successful deployment and adoption of something along these lines, it seems to me that several basic principles ought to be observed in its design, development, and promotion:

- **Open Model.** After initial implementation on at least two platforms, the framework model should be articulated completely in a freely implementable specification document (no patents, license fees, or reuse restrictions; but with a conformance test requirement for using the framework name promotionally).

- **Cross-Platform.** An implementation of the framework on any given machine should (as a goal, at least) be able to play media (including scripts and component configuration setups) created for all other implementations on all other machines, irrespective of host platform (although the component and framework binaries will necessarily be platform-specific, ditto any media that uses the unique features of any platform-specific component).

- **Vendor Independent.** Although private interests are expected to implement conforming components and frameworks as products, the model spec should not be the IP of any company or group of companies. Anyone should be able to write components, and all components should be able to communicate with each other.

- **Open API per platform.** An IA-SIG working group should establish the standard implementation details for the framework on each platform, to be similarly published, freely implementable, and not privately owned. This includes the specifying the

framework's API and the required interface suites for the components (i.e., list of function signatures to be supported).

- **Language-Independent.** Although the runtime interface and the framework object model's binary expressions will frequently talk in terms of objects, none of this should be considered as uniquely representable in terms of any particular programming language. Obviously object-oriented high-level languages will have an implementation edge here, but conforming API's could be implemented using (for example) table-slots instead of allocated objects in the event that language or other implementation issues so dictate. (Alternatively, object-oriented languages could be used in a wrapper layer around an existing non-object-oriented low-level audio API, to manage component instances and any objects internal to the component instances).

- **No Source Code Dependencies Between Components.** SOM may have an advantage over COM here.

- **Open Source Implementation per platform where possible.** Example/reference code implementing each platform's framework manager API and a small suite of example component shells should be equally freely available; ideally, some production-quality code would also be published.

- **Rely on Existing Open Standards where possible.** For obvious reasons including quick start-up and ensuring permanent viability, the framework implementation standard for each platform should make use of existing open technology standards wherever appropriate and possible– for example, in choices of supporting object, process, and communication models and technologies. The more low-level code that needs to be written anew, the greater the ongoing maintenance burden for our little community.

- **Support Generalized Authoring Environments where possible.** When components have a way of unambiguously representing their interfaces to the manager and to other components at runtime, then it becomes possible to create authoring environments that automatically adapt to the loaded components (see MAX, for example). This could go a long way towards solving our perennial problem of authoring tool development lagging API releases by years (or never coming out at all). The importance of this aspect should not be underestimated.

For Further Reading: *Component Software: Beyond Object-Oriented Programming*, Clemens Szyperski; 1997, 1998 Addison-Wesley; ISBN 0-201-17888-5.

..end..

```
****************************************************************************
```
.

Interactive Composition Column 1.2

The Eidos Interview
by Alexander Brandon

Introduction

In our last issue we introduced the topic of Interactive Composition through an examination of DirectMusic several key issues surrounding its introduction. Notably, the questions were asked "is DirectMusic too late?", and "just how can it differ from streamed audio?", The discussion focused mostly on theoretical issues surrounding the concepts of linear vs. non-linear audio.

In this issue we transition from the theoretical to the practical for an interview of members of the Eidos USA 'Soul Reaver' sound team by Alex Brandon (AB). Soul Reaver is the anxiously awaited sequel to the popular Blood Omen, Legacy of Kain. The team members interviewed include Jim Hedges (JH), Kurt Harland (KH), (of Information Society fame), and Amy Hennig (AH). *<Ed Note: In the following text "Legacy of Kain: Soul Reaver" will be referred to simply as SR.>*

Afterwards, we also asked Jim some additional questions about other Eidos projects which use their in-house Adaptive Audio system.

An Interview With The 'Soul Reaver' Sound Team

AB   Please describe the roles that the various team member playing in realizing the sound design for
       Soul Reaver and any other Eidos products that you want to talk about.

JH   For "Soul Reaver", the music was composed by Kurt Harland, while I did the adaptive audio
       programming, as well as being the "technical/creative liaison" between him and Amy Henig, the
       project's producer/director. For "Akuji", I wrote the music and did the adaptive audio
       programming.

AB   First, could you describe the first game in the series briefly and mention a few details such as what
       kind of response "Legacy of Kain: Blood Omen" received, and what building blocks you had to use
       for its sequel, such as the extensive use of voice in the original.

AH   Blood Omen: Legacy of Kain (the first game) was an action/adventure game in the Zelda tradition
       (not a hardcore RPG), with an overhead camera perspective and 2D graphics. The game was well
       received by critics and consumers, primarily for its unique theme and story line, which revolved
       around a "lone wolf" type hero afflicted with vampirism, and bent on revenge.

       When we set about to design the sequel, we knew we wanted to take the franchise into 3D, with a
       third-person camera, but retain the spirit of the gameplay-in the same way that Zelda64 was an
       evolutionary step beyond Zelda on the SNES. We also determined from the beginning to establish
       a new central character. Since Kain had essentially established himself as a dark god, ruling over
       Nosgoth, at the end of the first game, we thought it would be interesting to turn the tables, and
       make him the nemesis for a new protagonist.

       The quality of the story line, writing and the voice acting are distinctive elements of the Kain
       franchise-as opposed to similar action/adventure games, where plot is less important, and either
       text or second-rate voice acting is used to convey the story. It was important to us to retain the best

voice talent (most of the actors are reprising their roles from the first game) and directors-Gordon Hunt directed the sessions, and Kris Zimmerman (who directed Metal Gear Solid) is our casting director. There's roughly an hour of voice-over in the game, and about 100 in-game cinematic events.

AB   Describe, using technical as well as layman terms, the system you used for music in SR. Give a brief example of a part of the game where the music is used to enhance gameplay.

JH   We used an in-house developed adaptive audio MIDI driver, which replaces the Sony driver entirely. Signals from the game, based on location, proximity and game-state set special music variables, which are read by the driver and used to effect changes in the MIDI data. How these signals are interpreted is controlled by an extensive scripting language with standard branching, logic and arithmetic functions. This scripting language is written using MIDI text 'meta' events. These text commands can be written in a standard text file, or interspersed with other MIDI data in the MIDI bytestream. Some of the changes to MIDI data available are: muting/unmuting, transposition, pitch mapping, sequence start/stop, volume/tempo/pan changes etc.

As an example, in Soul Reaver, every piece of music in the game has several arrangements which correspond to different game states. The default arrangement, or "ambient" mode, is used when no signals from the game are present. When the player comes within range of an enemy (weather seen or not), a signal is sent to the driver which sets a designated "danger" variable. The script sees this change in the variable and mutes/unmutes tracks to produce a more intense "danger" arrangement. When the player engages in combat, another variable signifying combat is sent, and the same process ensues, this time with a tempo increase. If the player stops fighting or kills the enemy, the combat variable changes again, and this time certain tracks from the "combat" arrangement begin to fade out. If the player resumes combat, they fade back in. If no combat resumes, the combat tracks fade out entirely and the music changes back to either "ambient" or "danger" mode.

AB   What kind of leverage did you have in creating the soundtrack for SR? Did you control or at least coordinate the style of music? Were you able to recommend sound systems or request new features?

JH   The director of the game had some fairly well developed ideas regarding the music, however she chose the composer, Kurt Harland, specifically because she liked his style of music and thought it would be within his stylistic range. Since the arrangement of the music was so dependent on the interactivity, and the abilities of the driver, I had a lot of input into how the music was put together, since I would ultimately be responsible for making it "work" in the game.

I'm always able to request new features because the guy who wrote the driver, Fred Mack, works next door to me. I'm a constant pest.

AB   Did you collaborate cooperatively with the sound effects person(s) on SR, or did the two teams work mostly seperated?

JH   We did all the SFX in-house, so it was very collaborative. Sometimes we used the adaptive audio tools to create MIDI sound effect sequences which could not be created otherwise.

AB   Did you collaborate cooperatively with the level designers?

JH   Sound effects: yes, music: no. The music interactivity in Soul Reaver was specified on a very global basis. Most of the signals being sent were from the game code, as opposed to the level description files, so I worked mainly with the programmers. In the cases where signals needed to be sent from specific levels, I went in an edited the files myself.

AB   What kind of preparation did you do on planning the soundtrack, and what tools did you use to create it?

KH   I spoke with the producer, Amy Hennig, quite a bit and got her ideas on what sort of feel to give the game through the soundtrack. She not only described the interactive structure she had in mind, but also the ideas of the environments and characters in the game. For any given area, we took the history and nature of the creatures living there as the first inspiration for the soundtrack. For example, one of the regions of the game was inhabited by a race of mechanical-engineering-oriented vampires. Based on their goals and behaviors and on the intended smoky, mechanical environment in which they lived, I composed sounds and music which were thick, slow, and thumping, like big machines far away.

        Amy also gave me a lot of architectural drawings and photos to get a feel for the look of the environments.

        I used my very old computer sequencer: Voyetra Sequencer Plus Gold for composing, and Sonic Foundry's Sound Forge for designing sounds, mainly. Other than that, I primarily had to use the Sony tools for developing on the PlayStation.

AB   Was there anything particularly satisfying (or dissatisfying) about composing music for SR?

KH   The best thing was finally working on a game in which the music could be quiet, unobtrusive (except during combat) and filled with environmental sounds. I used rain, birds, screams, etc. This came at the request of the producer, and it's also always been the way I thought game scores could be more often. As has been said so many times, a fully-structured pop song can sound great to listen to once, but become a blood-sucking leech with teeth in your ear after the 47th repetition. This music is much less structured and "in-your-face". I loved getting a chance to do that after thinking about it for so long.

        Also, to combat this problem, Amy described a structure for the music tracks which would allow for the music to change a lot depending on what was going on. In this game, as you enter areas where enemies are known to be, the music gets a little more suspenseful. As you get within visual range of enemies, it sounds dangerous. And when you actually start to fight them, it becomes quite intense. The music also changes when you're underwater, outdoors vs. indoors, etc. This was a lot of fun artistically, even though it was a bit trying technically. It produced a lot of problems for composition, since it doesn't fit well with many aspects of music that people are used to and expect to hear. But it worked out quite well in the end.

        This interactivity was made possible not only by the creative vision of the producer, but also by some of the past and present audio department at Crystal Dynamics/Eidos USA: Fred Mack, Mark Miller and Jim Hedges. They developed their own interactive audio driver which makes all this possible.

        Dissatisfying? Well, the details would be a little boring, but in general, the biggest thorn in my side was making separate versions of all the music for use when the player is in the spirit (spectral) world instead of the physical (material) world. It caused numerous niggling problems that were hard to work out for everyone. But we did.

An Interview With Jim Hedges

AB   Describe, using technical as well as layman terms, the system you used for music in "Akuji the Heartless" (ATK, for short)

JH   The system was basically the same one that we used for SR. The music in Akuji was arranged in sections which corresponded to different sections of a given level. In this way the music progressed and developed on a large scale in accordance with the development and action of the game. To achieve this, signals would be placed at key points throughout the levels (entering a new room, fighting a battle, solving a puzzle etc.) These signals would trigger new sections of the piece to

play, so that the music would follow the level. These large scale sections were then broken down into subsections, which consisted of various tracks which could be muted and unmuted based on various game states.

For example: The player begins the level. The first part of the piece starts, playing the theme and setting the tone for the level. This section repeats until the player enters the second room, which contains enemies. Upon entering the room, a signal is sent to the driver, unmuting tracks which play the battle music for that section. The door closes behind the player, so they must defeat all enemies in order to go on. The battle music plays until the last enemy is defeated, at which point a signal is sent, the battle tracks stop, and the music goes into a new "post battle" section, which also serves as a musical transition to the next section. The door opens, allowing the player to leave the room and continue through the level. When the player leaves the room, a signal is triggered and the music moves on to the next section.

AB   What kind of leverage did you have in creating the soundtrack for AH? Did you control or at least coordinate the style of music?

JH   I had pretty much complete authority in writing the music. Once some basic styles were established (tribal, voodoo, heavy ambiance), I just ran with it.

AB   In working with the ATH team, how interested was the team as a whole in an adaptive audio system? Did you call most of the shots or did you work cooperatively in suggesting ideas? Did you use any techniques from your previous games such as the "Gex" series?

JH   Initially, the team wasn't very interested simply because they weren't aware of what an adaptive audio system could do. I had to sell the idea to the producer and lead designer. Once I started writing the music and implementing it, however, the whole team really liked it and were very supportive. I ended up calling most of the shots, but was helped greatly by the cooperation from the designers. I was constantly asking them to add new signals to their levels.

Gex II was the first game to use the adaptive audio driver, and used it in a limited way. Akuji was the first game to use the capabilities of the driver as a starting point for composing the music, so all of the music was written specifically for the driver.

AB   Did you collaborate cooperatively with the level designers?

JH   Yes, very much so.

AB   What kind of preparation did you do on planning the soundtrack, and what tools did you use to create it?

JH   Instead of sitting down and writing a self contained, complete piece of music, I started by writing in fragments. For example, a rhythm chart with different parts which could be muted and unmuted to produce different densities. Or an ambient bed which could support many different parts. In general, I composed different parts which could be combined in many ways. All of this was done using Studio Vision Pro along with various samplers and synthesizers. I then got together with the lead designer and went through the material with him, showing the various combinations possible. He would give me feedback on what combinations might work in certain areas of the game.

Then I would get together with the level designer and have them show me their level from beginning to end, so that I could get a sense of the structure of the level, including all important battles, puzzles etc. This structure would largely dictate the form the final composition would take. After the designer put in the signals for the level, I would go about scoring it, writing the script which would control the behavior of the music.

AB   Was there anything particularly satisfying (or dissatisfying) about composing music for ATH?

JH   The most satisfying thing was creating a complete adaptive soundtrack for a game from beginning to end, instead of slapping some interactivity on after the fact. The feeling I got when I first started implementing the music, and having it react to the gameplay, was the same as when I scored my first film.

AB   What games in the past that you have written music for have used adaptive/interactive audio?

JH   Gex II, Gex III (PSX), Mr. Bones (Sega Saturn), Tazmania, Taz II: Escape From Mars (Sega Genesis)

AB   Has there been a steady trend from your perspective either towards or away from the use of adaptive audio in games?

JH   I think it's growing, but it's not steady. Producers still need to be educated as to it's benefits. It will take a breakthrough title with great adaptive audio for the rest of the world to really notice.

AB   What techniques have you used, with what engines, for older title soundtracks, whether it be adaptive or not.

JH   Back in the day, I used the GEMS driver for the Genesis. That was a great driver for it's time, and had some adaptive audio features as well. When I moved on to the Saturn, I had to use the standard Saturn driver from Japan. Glad those days are over.

AB   How much control have you had in the past over your soundtracks?

JH   I've pretty much done what I want, weather people liked it or not In the past when I was a contractor, I would often not have control over the final mix. Now I make sure I always do.

AB   What kind of relationship do you have with the manufacturers and developers of the tools you use? For example, have you been able to suggest modifications to audio engines on the consoles you have worked on as well as PC based titles?

JH   I'm really fortunate to be working in a company with an audio programmer. He's the same guy who writes the tools, and I'm the guy who uses them, so I have a lot of input.

AB   What new techniques do you see being used in interactive audio? Is there a game or are there games in particular that you have seen use interactive audio in a new and effective way?

JH   I think the whole thing is still in it's infancy. I think the games coming out of Sony's 989 studios are doing great things with adaptive audio. Buzz Burrowes has a great driver there.

AB   Do you feel the average gamer can get more out of a game with an interactive score as opposed to looped or single shot playback?

JH   Absolutely. In the best case they will get an experience that is better than some movies. In the worst case they'll get music they won't be compelled to turn off.

AB   Do you feel that producers and lead programmers are considering interactive audio more important in games you have worked on? If not, what do the team leaders on games you have worked on prefer, and why?

JH   I don't think they consider it more important, they just agree to it because I bug them about it and promise them that I'll take most of the burden. With some exceptions, in my experience most producers and programmers prefer to not think about music until they absolutely have to, and then it's a little late for adaptive audio.

****

Section V. INDUSTRY CORNER
===========================

The bold industry is what drives development, and development drives the products. Paying close attention to the industry has been a very important duty for the IA-SIG. Read on and find out just what the blazes people out there are doing with your tools, both in software and hardware. *<Ed. Note: Items in the Industry Corner section are provided as useful information but do not necessarily represent the views of the IA-SIG or its management.>*

"Sound Engine Roundup" (Part 2: 3D APIs)
by Alexander Brandon

In our last issue we took a look at some of the engines in use today for music in interactive media. With this issue we begin to examine 3D sound APIs.

Although 3D sound and multi channel audio can be considered to be relatively new developments in PC based audio, they have certainly generated a lot of interest from the end user. Many game players are scrambling to purchase multi speaker systems and are enjoying playing their favorite new titles with headphones, awash in sounds that appear to come all corners of the room. This experience is helped to broaden knowledge of 3D audio and multi channel sound delivery formats, but it is also urging users to ask some questions.

For example, in Chris Grigg's 1999 GDC Interactive Composition Roundtable, 3D audio was seen by many content creators as an interesting prospect, but as something that was more in the hands of programmers than the sound designers. If you are a programmer, when you look at an audio API for your game, ask yourself as well as your API provider: "how easy will it be for my sound designers to use these features in our project?" If you are a sound designer or composer, ask the same question.

In this section, we offer comments from two providers of today's most advanced 3D audio technology: Scott Willing from QSound Labs, and Brian Schmidt of Microsoft.

QSound

In looking over QSound's Web Site and their product literature, we found the following two quotes which should serve as a decent introduction to their technology.

> "QSound signal processing algorithms provide positional 3D audio effects (Q3D tm) and soundfield enhancement (QXpander tm, Q2Xpander tm) through conventional stereo (i.e. two channel) sound systems. QSound also has algorithms for more advanced applications as well."

> "QSound Labs has been involved in interactive entertainment for nearly a decade. The tools and technology developed for interactive entertainment are also directly applicable or easily adapted to a wide range of applications including virtual reality, training simulators and communications."

From here, we contacted Scott Willing (scott.willing@qsound.com) and following is a question-answer session, in which he discusses the terminologies and misconceptions of 3D and multi channel sound, and how QSound delivers both technologies.

> *Q:* How does 3D sound differ from "surround sound"?

A:　There is often much debate over audio terminology, typically tainted by marketing angles. I'll try to give you my view and place it in the context of the other main view I've heard expressed.

Let's agree on a general goal: in the context of artificial sound (re)production using electronics and electric-to-acoustic transducers, we want to be able to synthesize the apparent spatial location of sound sources in arbitrary positions relative to the listener - in the ideal case, without restriction - in order to mimic the real world. [AB note: In my opinion, nobody has achieved the ideal case in a practical consumer technology-or even the impractical technologies I've heard. ;-)]

There are two general methods that can be used to approach this goal:

- use more transducers, driven by more discrete audio channels in more real physical locations, or,
- apply signal-processing techniques that extend the range of effective placement provided by a
    given number of transducers.

As compared to the two-channel stereo case, by my convention the former approach is termed "surround" and the latter is "3D audio". To put it in painfully simple terms, surround means more channels of output; 3D means adding signal processing to the channels you have.

Note, by my definition:
- the two technologies are not mutually exclusive by any means. That is, 3D techniques can be
    combined with surround techniques, as indeed they are in real products.
- soundfield enhancement (stereo expansion or surround enhancement) and surround
    "virtualization" are included along with positional 3D technologies under the umbrella of
    "3D audio".

At this point it's useful to examine another view of what constitutes "3D audio". There is a school of thought that suggests the term "3D audio" should be reserved exclusively for positional audio synthesis, and specifically, for positional audio based on the binaural synthesis technique: by nature, a two-output process. There have been impassioned arguments over this contention, which in my view is really a waste of time.

First let's deal with the positional aspect. I think it's widely accepted that the term "positional" audio has become synonymous with processing individual audio streams in an effort to place them independently in arbitrary apparent spatial positions defined in terms of azimuth, elevation and range (x,y,z) with respect to the listener. I can see where some would say, "if it ain't (x,y,z) it can't be 3D". The term "positional audio" would therefore apparently exclude soundfield enhancement, which typically operates on a mix rather than individual sounds, and involves no elevation or range-related processing. By extension, soundfield enhancement is also excluded from the definition of "3D" under this convention, which is at odds with my definition.

As long as everyone understands what the terms of reference are, IMHO it doesn't matter exactly where you draw the line. However, when people (such as vendors of 3D technology who lack soundfield enhancement in their product line-up) get on a high horse about "true 3D" I like to raise the following points:

The meaning of the term "stereophonic" has evolved in the collective conciousness to mean "two-channel". However, formed from the Greek word "stereos" (solid), it was originally intended to mean "solid sound" i.e. "sound having three-dimensional qualities". (Tongue stuck out) Bleah! So there! Even plain stereo is "3D". ;-)

The other bone I have to pick with the "true 3D" camp is that a given process that accepts a mono input and a (x,y,z) control inputs doesn't necessarily succeed in convincingly placing the sound at (x,y,z). Should it earn the term "3D audio" if it isn't fully effective, for a broad range of listeners, under representative real-world conditions? (Note: If it doesn't work on my desktop, but theoretically might work, or work better, in an anechoic chamber with my head nailed to a post, I

don't really care.) This is where the issue of binaural synthesis and its effectiveness should be examined, but I'll table that until later.

A further complication in defining 3D, especially as distinct from surround, is created by the substantial differences between interactive and passive audio. Generally I prefer to restrict the use of "surround" to refer to multi-channel (i.e. >2-channel) passive delivery formats and their associated gear. Dolby Surround, Dolby Digital, multi-channel MPEG etc. are intended for the delivery of a small number of streams that are separated out by a decoder and fed to multiple physical transducers. At production time, the content of these streams is pre-ordained as they are aimed at passive listening. They almost universally represent a mix of different sources, and they are intended to be directed to the listener from specific fixed physical locations.

But what then do we call the use of multiple speakers in an interactive environment? Multi-speaker 3D? (By my definition that would only apply if it involved filtering.) Interactive surround? Real-time surround? Multi-channel interactive sound, perhaps? Another debate.

In an interactive scenario, as I'm sure you're aware, it is common to abstract the actual method of sound placement in the interface used by the programmer. Microsoft's DirectSound3D API is the perfect example. Anyone can write an application that says "place this sound here" using DS3D commands.

It is up to the hardware how it attempts to achieve this. It may employ binaural synthesis and headphones, or binaural synthesis corrected for speaker playback and two speakers, or QSound's 2-speaker algorithm, or a QSound "3D+surround" multi-speaker algorithm, or it might just use "panning" (i.e. output channel volume ratios) and 600 speakers! The game is using "3D sound" in the sense that it is passing requests for sound positioning that, directly or indirectly, specify the position of various objects in (x,y,z) coordinates. The sound card that interprets this request using 600 speakers would technically not be employing 3D technology, yet it is compatible with a 3D API standard and might well be the most effective (if not the most practical) of the lot.

Q: If Qsound uses two speakers only, how does it claim to make sounds appear behind the user?

A: First, we don't use "two speakers only" but let's consider the two-speaker case. Actually QSound is the only 3D company that doesn't make this claim. I personally consider it The Big Lie of 3D audio.

We have been mercilessly slammed by competitors because our audio preprocessing tools (the QSystem, various plug-in's, QCreator etc.) do NOT provide a full 3D user interface. At this time, they're all designed for two-speaker output, and by their nature are intended for the production of canned content. (The reason for this statement will shortly become clear.)

It has been our opinion from the get-go that 2-speaker 3D (by whatever method) is primarily effective at placing sounds in a wide arc in front of the listener which extends well past the speakers. Though this is a dramatic, valuable, and exciting improvement over plain stereo production, rear placement effects and elevation effects are weak at best. Thus we have excluded rear placement and elevation from our user interfaces in such products. If we ever produce 3D preprocessing tools that support headphone 3D or multi-channel output, this will have to change.

I have never experienced any two-speaker 3D technology that I would present to someone, for example, like Pink Floyd mixer James Guthrie, and try to convince this professional audio engineer that I can put a guitar on the wall directly behind his head or stick a violin under the chair. Yet many 3D vendors are only too happy to make this sort of claim to sound-card buyers. How do they intend to get away with this? In my opinion there are a couple of issues.

The first is that these claims rely on the comparative inexperience and lack of professional-level discriminating audio savvy of typical sound card buyers. And for a while, it worked. But though

these folks may not be record mixers, they aren't all deaf or stupid. Pretty soon we started to see threads pop up in newsgroups with subject lines like: "3D Audio, Is it a Joke?" This really pissed us off, because the unnecessary and insupportable marketing claims made everyone in the business look bad.

The other issue is that the perceived effectiveness of any 3D audio process is dramatically extended by creative use, and in typical interactive applications, all the methods are available: Interactivity itself, supporting visual cues, preconditioning and motion (often lots of it) can all add up to a very convincing net illusion. There's nothing wrong with this per se - that's where the art comes in - but credit where credit's due, please. So, a good demo can lead to the impression of capabilities that aren't really there.

So: Turn off the video monitor, turn out the lights, take the interface away from the user and place arbitrary, unfamiliar sounds in arbitrary static locations. Ask the listener identify their positions. An interesting story comes to light if it hasn't already.

Repeat the test with headphones. You'll find that binaural synthesis, the darling of 3D "purists", fails miserably at creating convincing placement in the front hemisphere. (Only one well-known 3D purist I know of has admitted this in print - though the same fellow once told us 3D over speakers was "impossible" but now heads the 3D audio department at a Big Company that sells A Lot of Speakers. <g>)

Add crosstalk cancellation to the same binaural filtering algorithm, thus theoretically adapting it for speaker playback, and the resulting output displays the characteristic weaknesses of speaker algorithms: front placement is fine, but elevation (which can work quite well on headphones) now sounds like tonal variations - i.e. "flavor" rather than effect, and rear placement... just isn't there.

Near the beginning of my post I told you I don't personally think anyone has come close to the ideal of completely successful arbitrary sound placement synthesis. I'll stand by that contention until proven otherwise.

[Sidebar: In an interactive scenario, one MUST provide a full 3D API to the application programmer that assumes no rendering engine limitations. You can't tell the programmer that it's impossible for the enemy battle cruiser to be behind and above the listener. In turn, the engine must do the best it can (logical behavior at least) with whatever algo's and output configuration it supports to attempt to produce an appropriate result. Another reason the API should assume no limitations is because the user has a selection of run-time rendering engines, and some are better at certain aspects of sound placement than others.]

Q: Does QSound have more than 2 speaker technology available or in development?

A: Yes. First of all, there's the case of surround formats. Our QSurround process has combined 3D with multi-channel formats for some time, both for "virtualization" (surround rendering over two speakers or headphones) and for surround enhancement (multiple physical output channels + 3D processing).

For interactive apps, there are lots of four-speaker sound cards out there now; next-gen sound cards will very typically be sporting "5.1" output features. Our first positional (Q3D) 4-channel implementations used simple panning (not 3D) but that's all changed now.

[Sidebar: I personally think that a better arrangement for interactive multi-channel audio via six outputs would be "2/2/2" rather than "3/2" + sub. (That is: 2 front / 2 side / 2 back vs L,R,C, Ls, Rs + sub) However, since people will apparently be using their PC's to watch canned surround content authored to the 5.1 format (?) I guess we're kind of stuck with that arrangement in the interest of compatibility.]

Q: Would you, as an unbiased opinion, consider QSound or A3D to be a more popular choice for game developers?

A: In a very real sense, it's impossible to make an easy comparison - these are apples and oranges. First, bear in mind that you have to separate the idea of the API from that of the rendering engine. (This is where my FAQ has a lot of relevant info.) Our philosophy with respect to 3D sound hardware on the PC has been that we support industry-standard, open API's. That means all sound cards employing Q3D may be controlled entirely via Microsoft's DS3D API for positional 3D, and the EAX 1.0 DS3D property set extension for reverberation. We are currently adding support for the I3DL2 guidelines, which pick up where EAX 1.x left off and, as you may know, are loosely based on EAX 2.0.

Bottom line: to support our hardware, the developer does NOT have to use a QSound API. They can use anything that supports DS3D / EAX protocols, including third-party SDK from folks like RAD (Miles Sound System) etc. The idea of a proprietary API for hardware is ostensibly to provide support for proprietary features that an open API like DS3D doesn't address. In fact, DS3D has a mechanism called property sets for this very purpose. (Like MIDI Sysex.) In fact, the real idea of a proprietary API is to claim titles that support the API as your own. This is purely a marketing exercise.

QSound also has software development kits, and thus our own API. Why? Bear in mind that QSound was providing real-time 3D capabilities back in DOS days. (Terra Nova: Strike Force Centauri, for example.) Our QMixer API was born under Windows 3.1, believe it or not. There WAS no hardware to support, no DS3D, no Win 95... only our Q3D algorithms running in software. (First QMixer title: Zork Nemesis by Activision.)

Some developers consider 3D audio to be an important-enough component of their titles to want to ship functional software 3D that will work with any stereo sound card. We provide this in the form of the QMixer SDK. Since QMixer contains our patented 3D software engine, we charge a significant license fee for it.

I think that willingness to belly up and pay good money for a technology qualifies as a true endorsement, don't you?

You may have noticed that we have a free SDK as well, however. QMDX has the same high-level, full-featured API that QMixer does, but its internal engine only mixes to stereo in the absence of a hardware 3D accelerator. Both QMDX and QMixer use DS3D and (in the next version) EAX commands to use any manufacturer's compliant 3D hardware to render in 3D, with reverb, if the capabilities reside in the end-users' hardware. QMDX is offered for free, to underline our commitment to open API's and to assisting developers with supporting the broadest possible range of hardware.

We don't even ask developers to register with us to get QMDX. It's freely available for download. There's no logo requirement. We have no idea of how many dev's are using it.

Q: Some other 3D audio API's, like Creative's "Environmental Audio" use multi-speaker systems to achieve surround sound. This would seem to be a more effective way of getting genuine 3D / surround sound in games, and it seems to be working with the less expensive satellite speaker systems they are distributing. Would then the next step overall for 3D sound be to increase the number of speakers, or develop new technology other than the standard 2 speaker system to create 3d sound?

A: OK, let's first step back a bit. Some people think that headphones are the only way to go. I personally hate 'em. In my passive entertainment environment (living room) I would implement multi-speaker surround with QSurround enhancement, but on my PC set-up at home, multiple

speakers really aren't convenient. On the PC I would stick with two speakers, and I put on headphones only when I'm starting to piss my girlfriend off - or I need to test a headphone algo!

My point here is that there is room for many opinions, and for reasons of preference or practicality, there is no perfect solution for every user. The real beauty, IMHO, of interactive audio on the PC is the fact that rendering takes place in real-time, and there is a separation between API and rendering engine. The app doesn't have to care whether the output goes to headphones or 47 speakers or a freakin' bone implant. ;-)

If you want to compete in the PC sound card market, you have to give the user all the choices you can. I'll say it again: next-gen products from virtually (no pun intended) all manufacturers will have at least four and up to six outputs, but will or should support two-speaker, four-speaker and headphone 3D as well. In any set-up of up to "5.1" channels, the addition of 3D processing to the multiple output channels definitely adds value. This is probably an area that deserves further research by all companies, as the lion's share of attention in the 3D camp has obviously been focused on the assumption of two-channel output

.

(BTW, I think it's a bit funny to watch the purist camp adapt to the reality of public preference for multiple speakers. The pure binaural + crosstalk cancellation model is very much at odds with this output scenario, for one thing. Providing multi-speaker support also forces a tacit admission that two-speaker 3D isn't, at least in the public's opinion, up to the marketing claims that typically come with it. We've never played the "3D is perfect" tune, so we're totally comfortable with this situation - in fact we think it's a really healthy thing.) If manufacturers get into a "number of speakers" race, you quickly get to the point where the additional value that might theoretically be added by 3D processing to simple "volume ratio" signal distribution amongst output channels would be minimal. However, there's no technical reason why someone couldn't produce a sound card with a horrific number of outputs. It's pretty easy; the question is how many channels of D/A, and speakers to go with them, is the public prepared to buy?

<Ed. Note: Thanks to Scott for his effort in answering these questions as fully as anyone could expect!>

For more information contact:

    Scott Willing
    QSound (www.qsound.com)
    Suite 400, 3115-12 Street North East
    Calgary, AB.
    Canada T2E 7J2
    403.291.2492 (voice)
    403.250.1521 (fax)

Microsoft DirectSound3D

Next, we took a look at the DirectSound3D. The following passage found on Microsoft's DirectX web site ((http://www.microsoft.com/directx/overview/dsound3d/default.asp) will serve as our introduction.

    "The Microsoft® DirectX® audio playback services are designed to support the demanding requirements of games and interactive media applications for Microsoft Windows®. Microsoft DirectSound3D allows you to transcend the limits of ordinary left-right balancing by making it possible to position sounds anywhere in three-dimensional space. And it can be accelerated in hardware or implemented in software when 3-D hardware is not available."

We got in touch via email with Brian Schmidt, the Program Manager for both DirectSound & DirectMusic: He supplied us with the following comments:

"DirectSound3D is a neutral API. It merely specifies x,y,z locations of sounds along with being able to specify a directional sound (sound cones...for example, if you're facing away from me and talking, you'll be softer than if you were facing me and talking). It also lets you control the amount of Doppler effect and set the maximum and minimum distances for a sound. These aspects are almost 100% non-controversial.

The "DirectSound3D renderer" takes the position information from the API and creates the sound. If there is hardware available, the DS3D renderer is not used; DirectSound uses the hardware instead of doing its own processing.

DirectSound can also be extended to include processing that the hardware can do but the DirectSound3D renderer can not. A good example of this is EAX reverb. Suppose a game uses DirectSound3D; On cards that support 3D and EAX, the hardware gives the user 3D and EAX. If the card only supports 'Frank's 3D', the user gets 'Frank's 3D'. If the card doesn't support any 3D, the user gets DirectSound's 3D.The goal is that if a user has any 3D sound card, and a game wants 3D audio, the user should hear whatever their card can do (after all, they choose that card). So the API should be neutral, and not tied to any specific piece of hardware. This is in fact the essence of all the DirectX API's.

(As a side note: new in DirectX 7 is the ability of a programmer to choose from 3 different DirectSound 3D renderer algorithms; plain stereo-pan+doppler+volume, and 2 different HRTF algorithms licensed from Harman Industries, called 'VMAx'."

Next month, we hope to add additional information on 3D Audio Engines from other leading companies.

<Ed. Note: Thanks to Brian for his more in depth explanation of DirectSound3d, it being one of the most widely used 3D audio APIs in games today.>

For more information on DirectSound3D, contact:

Brian Schmidt
DirectSound/DirectMusic Program Manager
Corporate headquarters:
One Microsoft Way
Redmond, WA 98052-6399
Telephone: (425) 882-8080

****

Section VI. Developers Corner
========================
At last, a place where the developers can speak their minds. The interactive audio industry is bursting with men and women ready to both complain and cheer about various aspects of their work. Here is an outlet for them.

This issue we look at an area that hasn't been the subject of much IA-SIG activity yet, but is where gaming popularity got most of its boost in the 80s: The Arcade. While one might say that arcades are long past their heyday, many companies are still pushing the envelope and developing leading edge titles with far more sonic firepower than can be found on the average PC or anything else at home.

We will start by going back a little with Graeme Norgate, music and sfx wizard at "Free Radical Design, Ltd.", formerly attached to "Rare, Ltd.". Here, Graeme talks about his role in the widely popular "Killer Instinct":

"Throughout 94, whilst working at Rare myself and another musician worked on the audio for Killer Instinct which hit the arcades around Christmas the same year. The audio hardware was the same as used by Midway/Williams in most of their coin-ops and Pinball Machines at the time, most notably Mortal Kombat 2.

We had 4 meg in total for all sfx and music, but this was running mpeg compressed audio at pretty high compression rates up to 50 to 1 in some cases if my memory serves me well. This obviously caused a lot of bass bubbliness and high treble tweetering, but in an arcade environment this wasn't going to cause much trouble. To keep within the memory limit, however, the music was short and repetitive and the whole mix would be cut up into little blocks of sound that could be repeated and the original tune was then rebuilt with these samples by the use of a playlist.

There were no effects, and playback was only 3 channels. So music was in mono which left 1 sound channel for each character, you couldn't play looped samples, nor change the pitch.

It was fun to do though, as we could use full blown music mixes for the soundtrack which compared to the 8 channel SNES chip music we were also developing on at the time was a luxury.....although it came back to bite me in the backside when I had to down the SNES and gameboy conversions :)"

- Graeme Norgate, Free Radical Design

Next we hear about the ups and downs of development for Atari Games' 'Gauntlet Legends' from John Paul:

"Creating the music and sound for Atari Games' recent arcade release Gauntlet Legends was a game composer's dream and nightmare. The fun part was using the original Gauntet audio with its catchy FM organ tune and effects as a stepping stone for a sample-based update. And the new graphics produced by our artists were beautiful and evocative, great inspiration for 70-plus minutes of music and almost 1000 sound-bites. The challenge came in getting it all to work on our target custom hardware.

The first limitation (as always) was RAM: 2 Megs initially which was later doubled (may the engineering gods be praised!). Now I know to PSX developers that sounds like a lot, but when you consider this was a 4-player game, that most of the 300 announcer phrases always had to be available and that the music "streams" were also in RAM, it was a tight fit. A fair amount of time was spent in designing an efficient banking scheme, working with the programmers to determine when banks could be loaded from the game's hard-drive into RAM. In an arcade title you try your utmost to get those load-times to be almost imperceptible, so this was no trivial matter. We also have on-the-fly decompression so I was constantly tweaking the compression parameters to fit yet one more sound.

The second more troubling limitation was that we only had 6 channels of audio to play back everything. With 4 players each of whom could be engaged in individual battles with a myriad of monsters, ambient sounds and music vying for the 6 channels, it became a logistical nightmare to insure adequate game audio. We started working on a complex priority scheme and I had to be constantly on the programming staff to share the latest fighting engine rules so I could recommend adjustments to the scheme accordingly. (Priorities are set by the programmer but I was the only one who wanted to do the rock/paper/scissors type of envisioning that is required in priority planning.)

It became very clear mid-development that the sfx package was not going to work unless I starting stacking sounds on top of each other, creating single samples from an assortment of swooshes, punches, and grunt reactions. This approach meant creating a large number "combo" sounds which further taxed our RAM limitation. My ProTools system was a great tool in this

endeavor, easily allowing me to add new layers of sound to already constructed combos as our fighting matrix became more and more complex. I believe in the end that the effort was worth it, since it afforded the opportunity to have a more varied sound set than normal and created the illusion of it being more interactive than it really was.

With this much sound going on, balancing it all was key. Our system allows volumes to be set by the composer at the bank construction level and then adjusted by the programmer within game code. Though this approach can create havoc in debugging (programmers have little time to track down each volume call), it does have its advantages. Programmers usually group sound calls into groups (i.e. announcer voice, boss sfx, player sfx) so that volumes can be set globally making it easy to raise or lower a certain category within
the mix. It also gives the composer the ability to tweak each individual sound without bothering the programmer.

Designing audio for custom hardware can be a challenge. There were many times I would look at envy at some of the capabilities of the consumer platforms. Other times I would be thankful that I didn't have to worry about some of their restrictions. And with custom systems there is always the possibility that they improve. During the development of Gauntlet, audio RAM did double in size and we gained the ability to do stereo streams off the game's hard-drive. Both improvements drastically improved my ability to create successful audio for the game."
- John Paul, Atari Games

COMING IN THE NEXT ISSUE (November 1999)

- - "Big in Japan", interviews with some of the biggest names in titles from the mother country of video games: Japan.
- - "Multichannel"... the second part of our audio engine roundup, this will cover features of Dolby systems and other multichannel options for games.
- - "The Other Interactive Music"... G-vox, Koan, Reality, The Axe, even Dance Dance Revolution.. we've heard many titles that teach and let you create music 'interactively'. Time for an examination of what all the fuss is about. It will be included with an IMX report.